PTO: 2003-5395

Japanese Published Unexamined Patent Application (A) No. 11-143694, published May 28, 1999; Application Filing No. 09-312225, filed November 13, 1997; Inventor(s): Daisuke Yamada; Assignee: Ricoh Corporation; Japanese Title: Software Version-Up Method

---

SOFTWARE VERSION-UP METHOD

CLAIM(S)

1) A software version-up method for a system, wherein the software is divided into components, and multiple software components cooperate in operation, characterized in that each software is assigned with an identifier and its version data, and in that said identifier and said version data are together loaded when a new software component is loaded and put to version-up.

2) A software version-up method, as cited in Claim 1, wherein an assembly table and a combination table of multiple software components cooperating in operation are loaded when a new software component is loaded and put to version-up.

3) A software version-up method, as cited in Claim 2, wherein, when a new software component is loaded and put to version-up, it is selected out of a set of multiple software components based on the version data of each software component and its compatibility.

DETAILED DESCRIPTION OF THE INVENTION

(0001)

(Field of Industrial Application)

The present invention pertains to a software version-up method.

(0002)

(Prior Art)

Japanese Published Unexamined Patent Application 04-098529 discloses a program creation support system whereby a program created by a senior programmer is made usable by a junior programmer. Japanese Published Unexamined Patent Application 04-034772 discloses a method whereby a program component that has already been developed is automatically modified to be reused when a new program is developed. Also, Japanese Published Unexamined Patent Application 02-067629 discloses a method whereby multiple program components that have already been developed are retrieved to be reused when a new program is developed.

(0003)

(Problems of the Prior Art to Be Solved)

These prior arts, however, are the methods for reusing the program components already developed to improve the productivity when a new

program is developed, so the program that has been already incorporated in a product cannot be put to version-up, which is a problem.

(0004)

The present invention, taking the aforementioned problems into consideration, attempts to present a version-up method that allows to simply control the incorporated software part in the case when the software components are put to version-up in a system where multiple software components cooperate in operation.

(0005)

(Means to Solve the Problems)

To accomplish the aforementioned objective, the present invention presents a version-up method for software, wherein a first means makes the software into components, and multiple software components cooperate in the operation. The method is characterized in that an identifier and its version data are assigned to each software, and said identifier and said version data are both loaded when a new software component is loaded for version-up.

(0006)

When the first means loads the new software component for version-up, a second means loads the tables indicating assembling and combining of multiple software components cooperating in operation.

(0007)

When the first and second means load a new software component for version-up, the third means selects it out of multiple software components based on the version data of every software component and its compatibility.

(0008)

(Embodiment Examples)

The embodiment examples of the present invention are explained below with reference to the drawings. Fig. 1 shows a schematic diagram of the system that is used in the software version-up method, as one embodiment example of the present invention. Fig. 2 shows a block diagram of the image input/output device structure of Fig. 1. Fig. 3 illustrates the software structure of the image input/output device of Fig. 1. Fig. 4 illustrates one example of the software component of the image input/output device of Fig. 1. Fig. 5 shows the data structure of the software component. Fig. 6 illustrates one example of the data structure of the software component of Fig. 5. Fig. 7 illustrates the data structure of the assembly table. Fig. 8 illustrates one example of the assembly table of Fig. 7. Fig. 9 illustrates the data structure of

combination table. Fig. 10 illustrates one example of the combination table of Fig. 9. Fig. 11 shows a flowchart illustrating the selection algorism for the software components.

(0009)

As shown in Fig. 1, in the user environment 1, copier, printer, facsimile servers 2 and 3, scanner 4, and personal computer (PC) 5 are used, and these terminals are operated by the software. These image input/output devices, 2-4, such as the copier, the facsimile servers, and the printer, and PC 5 may be used in stand-alone state or by one-on-one connection, but in recent years, they are increasingly connected to the network environment 6-2.

(0010)

Also, some vendors install a service center to offer a service to the users and remotely help the users in operation and use method. To version-up the software of the image input/output devices, 2 – 4, PC 11 and memory 12, in which the software component is stored, are installed in the service center 10. This PC 11 is, via the network 6-11, connected to network 6-2, image input/output devices, 2 – 4, on the user's side, and loads the software component directly to the user's image input/output devices, 2 –5, by a remote down load method.

(0011)

The structures of the image input/output device 2 – 4 in the user environment 1 are explained with reference to Fig. 2. The processor (CPU) 21 controls all the image input/output devices, 2 – 4, and under this control, are connected ROM 22, RAM 23, NVRAM 24, operation panel 25 and panel controlling section 26, scanning/printing engine 27 and engine controlling section 28, memory device 29 and disk driver 30, communication controlling section 31, and modem 32.

(0012)

In the ROM 22, the program code, font, and other static data are preliminarily stored. The RAM 23 is used to temporarily store data, and NVRAM 24 is used to store volatile data. The operation panel 25 and panel control section 26 serve the interface connected to the user. The scanning/printing engine 27 and engine control section 28 read and print an original copy on the transfer paper as the image data input/output device. The memory device 29 and disk driver 30 are used to store a large volume of data and data base. The communication control section 31 is connected to the network 6-2, such as Ethernet, to communicate with the external devices. Modem 32 is used to communicate with external devices via the public line.

(0013)

Broker 40 maintains the function data and the control data necessary for image input/output devices, 2 – 4, connected to the network 6 – 2, and it is a middleware that establishes the connection between the client (PC 5 in this specification) and the server (image input/output devices, 2 – 4 here). This device 40 has CPU 41, ROM 42, RAM 43, communication control section 44, and data base 45. This device 40 may be positioned on the side of either PC 5 or image input/output devices, 2 – 4.

(0014)

The structure of the software for the image input/output devices 2 – 4 is explained below with reference to Fig. 3. The structure primarily consists of four layers: application layer 50, kernel layer 60, driver layer 70, and hardware layer 80. The application layer 50 forms the applications, such as those of copier, facsimile, and printer, and has operation manager 51, document manager 52, service manager 53, device manager 54, data manager 53, and program factory 56.

(0015)

The document manager 52 is a central function of the application that handles a document along a scenario of copying, faxing, and printing. The

service manager 53 is a necessary function as well when a document is handled, and it controls and carries out various services. The device manager 54 is a function that determines the operation of the physical device such as image bus 86, and controls and carries out various services. The operation manager 51 is a function that controls the operation panel 25 shown in Fig. 2, and displays buttons, notifies the button operation, and issues an alert notice.

(0016)

The data base manager 55 maintains and controls continuous data, such as font, fixed form, fax reception history, use history of the device, charged fee data, and it also, as explained later, maintains and controls the software components and their assembling. The program factory 56 initializes the software assembly table, software components, and their compatibility for program execution. More specifically, it prepares an operable condition by opening the static software component on the RAM 24 (generates an instance in object-directed programming) and by receiving some massage (method call in the object-directed programming).

(0017)

The kernel layer 60 is generally incorporated as the kernel of OS, and presents a service to the application 50 by abstracting various devices of imaginary machine 65 having imaginary memory 61, execution process 62,

file system 63, socket 64, execution control 65a, and mode control 65b. The application layer 50 operates by making a system call to the kernel layer 60. The driver layer 70 is a set of functions that drive, control, and carry out various hardware, such as memory control driver 71, process-control driver 72, file-control driver 73, process-control driver 72, file-control driver 73, network driver 74, integrated type copy driver 75, blocking-device driver 76, and page-device driver 77. The hardware layer 80 is a set of controllable resources that are present in the devices, such as FOM 2, RAM 23, NVRAM 24, network interface 84, scanner 27a, image bus 86, and blotter 27a.

(0018)

Fig. 4 shows, as one example of the software component of image input/output devices 2 – 4, an assembling steps of the software components incorporated in a copier and scanner to operate the carriage at a time of reading an original copy. The square boxes, 101 – 103, show the software components, respectively. The scanner control component 101 controls the carriage component 102 and is responsible for operational steps of the carriage according to the ranges of resolution and of reading. The carriage component 102 instructs the moving speed to the motor component 103 and is responsible for executing and monitoring the carriage operation. The motor

component 103 is a software component responsible for driving the stepping motor.

(0019)

Fig. 5 and Fig. 6 show the data structure 110 of the software component that consists of component ID 111, name 112, version 113, and of attribute value unique with individual component 14. The component ID 111 is a proper identifier of every software component that is preliminarily registered, and generally consists of 6 digits of numerical value (for example "012345" as shown in Fig. 6); a vendor who offers the device comprehensibly controls the use of the software component based on the identifier. Name 112 is the name of the software component and is generally indicated by properly selected sequence of characters(for example, "carriage" as shown in Fig. 6). Version 113 is the version corresponding to the component ID 111, and generally consists of 3 digits of numerical value. For example, as shown in Fig. 6, "101" indicates the version 1.01. Attribute value 114 unique with individual component is the data necessary for each software component.

(0020)

As shown in Fig. 7, the data structure of "assembly table" 120 is constructed by two components to be assembled, which are ID 121 of

component "1" and ID 122 of component "2," the pointer 123 for the data structure of "combination table" 130 of Fig. 9, and by "the subsequent combination table" 124. The subsequent "combination table "124 has a list structure for holding multiple sets of combination, and "NULL" indicates there is no "subsequent combination." This "subsequent combination table" 124 needs not to have a list structure as long as it takes a form of a collection structure that can contain multiple data. Fig. 8 shows one example of "assembly table" of scanner control component and carriage component.

(0021)

As shown in Fig. 9, the data structure of the "combination table" 130 contains version values, 132, 134, indicating the version of component "2" that can be combined with version values, 131, 133, indicating the versions of component "1" (Fig. 7) and with the values of the higher versions, 131, 133, and the data structure end with "NULL." This structure can only handle the combination having the upper order compatibility. But, by arranging this table two-dimensionally, the compatibility of all the versions between two components can be listed.

(0022)

In Fig. 10, as one example, "012345" is defined as the ID of the carriage component, and "121212" as the ID of motor component. Then, the

table shows that the component having 2.00 or below can be combined with the component of versions 1.50 or below, and that the carriage component having version 3.00 or higher can be combined with the motor component having version 1.80 or below. Accordingly, when there is only carriage version 2.00, the motor component of version 1.80 cannot be used, so the motor component in version 1.50 or below is used, so this prevents the version-up of motor alone.

(0023)

Registration of said software component unit is maintained and controlled as the software component data base by the data base manager 55. Fig. 11 shows the process of selecting the software component to be operated from the database of the registered software components. First, the "assembly table" shown in Fig. 7 and Fig. 8 is acquired (in step S1), and the ID 121 of the component (1) is searched (in step S2). Subsequently, whether the component "1" exists or not is determined (in step S3). If it exists, the manager moves to step S4 or below, and if it does not exist, the manager moves on to step S6 or below.

(0024)

In the steps S4 and below, the version values, 131, 133, of the component "1" are compared with the maximum value (step S4), and if they

are lower than the maximum value, the database manager goes back to step S2, and if they are higher than the maximum value, the version values, 131, 133, are kept as the maximum values (step S5). Then, the manager goes back to the step S2. In the step S6 or below, the ID 122 of the component "2" is searched (step S6), and the ID 122 of the component "2" exists or not is determined (step S7). If it exists, the manager moves on to step S8, and if it does not exist, it moves on to step S9 or below. In step S8, the version values, 132, 134, of the component '2" are sorted out in the descending order, and the manager goes back to step S6.

(0025)

In step S9 or below, a pointer is set at the head of the list of the components "2" in the descending order (step S9). Subsequently, whether the component "2" exists or not is determined (step S10). If it exists, the manager moves on to step S11 or below, and if it does not exists, the manager moves on to step S14 or below. In step S 11, whether the component "1" and the component '2" can be combined or not is determined. If they cannot be combined, the pointer of the list in the descending order is incremented in step S12, and the manager goes back to step S10. In step S 13, the component "1" and the component "2" are registered, and subsequently, whether there is the subsequent combination or not is determined (step S15).

If there is the subsequent combination, the manager goes back to step S2, and if there is no subsequent combination, the process is completed here. If there is no component "2" found in step S10, the version lower than component "1" by one version is searched in step S14, and ID 122 of component "2" is searched in step S6.

(0026)

Accordingly, by the above selection process, for example, in the case when one carriage component and one motor component are both put to version-up non-synchronously, more specifically, a combination of version 2.00 of one motor component and of version 1.80 of one carriage component does not operate, the older version of the motor component, for example, version 1.20, is used. In addition, if the version 3.00 of the carriage component is registered and can be combined with the version 1.80 of the motor component, the subsequent operation can use the version 1.80 of the motor component.

(0027)

(Advantage)

As explained above, according to the invention mentioned in Claim 1, in the case when each software component is put to version-up in a system, wherein the software is divided into components and multiple software

components cooperate in operation, each software component is assigned with an identifier and its version data, and the identifier and the version data are loaded together; therefore, the software component incorporated in every device can be detected and the software can be handled smoothly when the function needs to be expanded or the hardware needs to be exchanged. In addition, when a malfunction has occurred to the software, the cause can be easily detected, and the software component can be exchanged.

(0028)

According to the invention of Claim 2, when a new software is loaded in the system and put to version-up, a stop of the system operation due to version-up of one software component is prevented since the assembly table and the combination table of multiple software components cooperating in operation are loaded.

(0029)

According to the invention of Claim 3, when a new software component is loaded and put to version-up, it is selected out of a set of multiple software components based on the version data of each software component and its compatibility, so the software component can be handled easily when the hardware is exchanged and it can be exchanged from the service center via a network.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 shows a schematic diagram of the system used for the version-up method as one embodiment example of the present invention. Fig. 2 shows a block diagram of the software of the image input/output device of Fig. 1. Fig. 3 illustrates the structure of the software of the image input/output device of Fig. 1. Fig. 4 illustrates one example of the software component of the image input/output device of Fig. 1. Fig. 5 illustrates the data structure of the software component. Fig. 6 illustrates one example of the data structure of the software component of Fig. 5. Fig. 7 illustrates the data structure of the "assembly table." Fig. 8 shows one example of "assembly table." Fig. 9 shows the data structure of "combination table." Fig. 10 shows a flowchart for explaining the selection algorithm for the software component.

111. component ID (identifier)

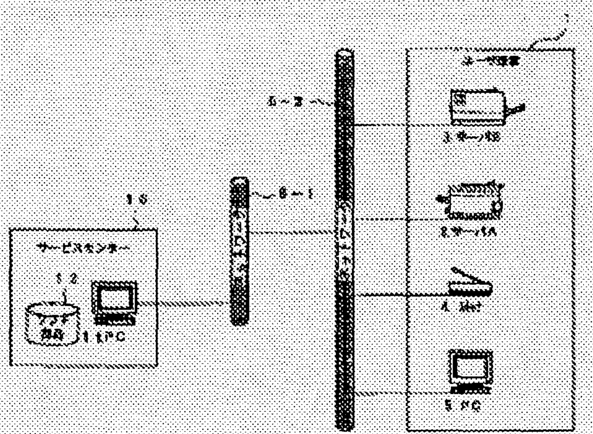
112. component name

113. version

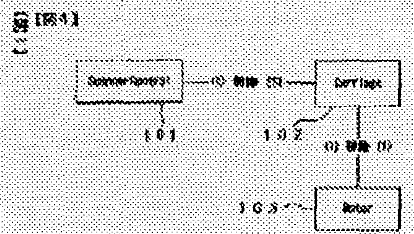
120. assembly table

130. combination table

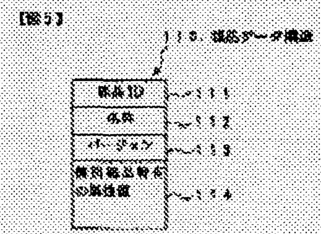
【図1】



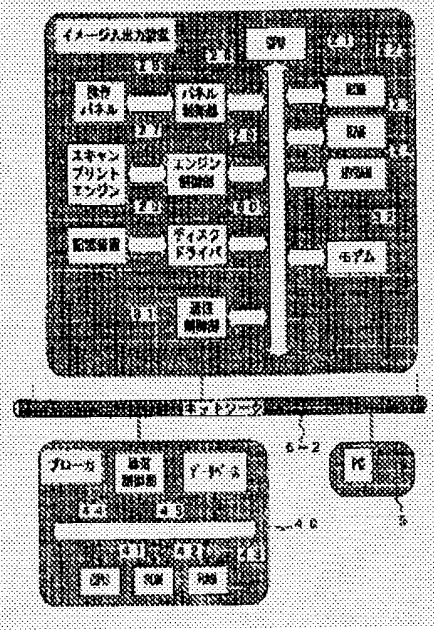
【図4】



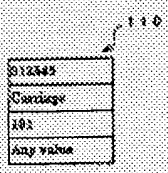
【図5】



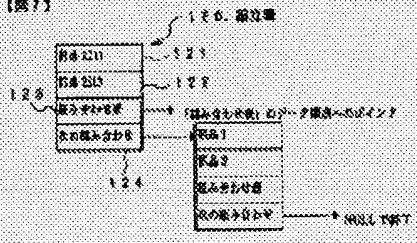
【図２】



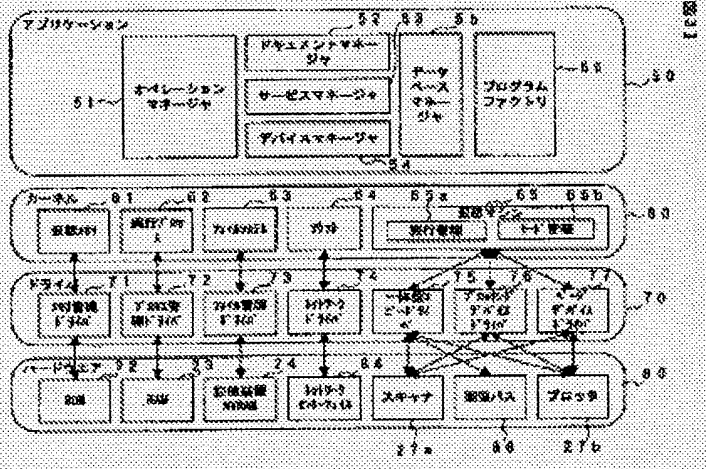
【図６】



【図７】

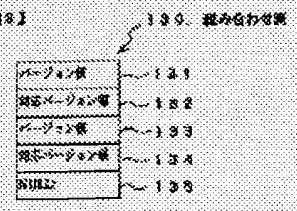


【図３】



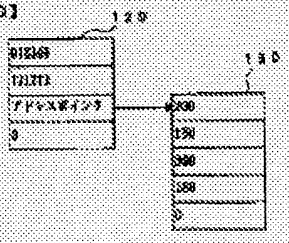
【図９】

【図８】

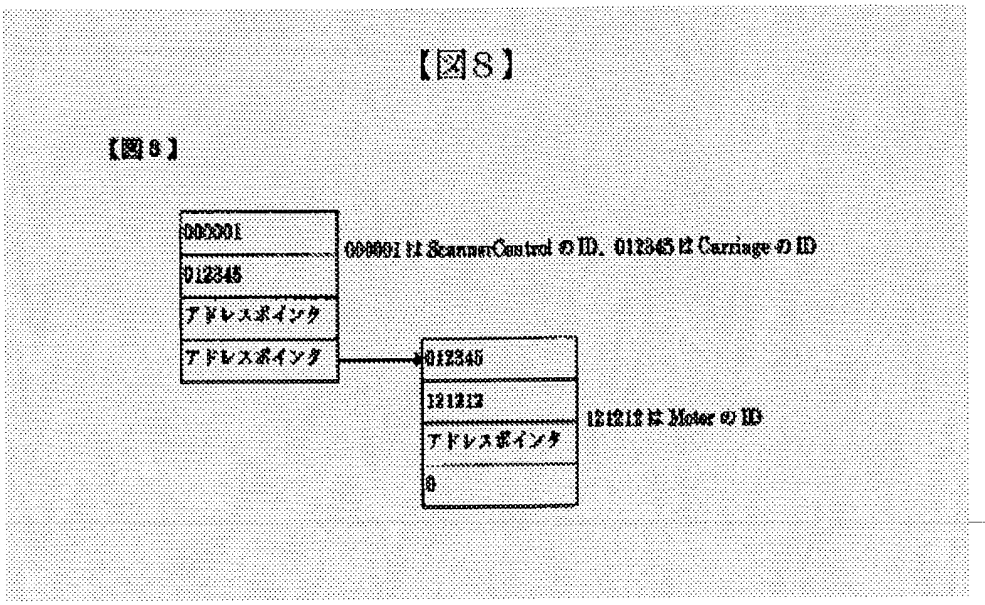


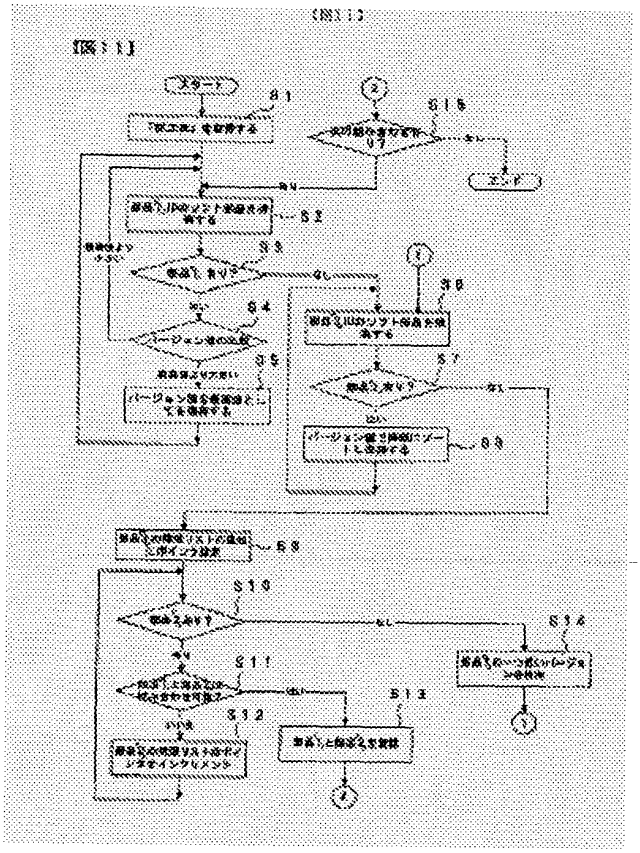
【図１０】

【図１０】



【図８】


【図８】
000001
012345
アドレスポインタ
アドレスポインタ
000001 は ScannerControl の ID、012345 は Carriage の ID
012345
121212
アドレスポインタ
0
121212 は Motor の ID



Translations
U. S. Patent and Trademark Office
9/11/03
Akiko Smith